VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2019 Anna Dzuňová

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Anna Dzuňová**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Projektově.CZ s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Lenka Skanderová, Ph.D.**

Konzultant bakalářské práce: Ing. Zdeněk Solnický

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prehlasujem, že som túto bakalársku prácu vypracovala samostatne. Uviedla som všetky literárne pramene a publikácie, z ktorých som čerpala.

V Ostrave 15. apríla 2019 Dzuňová .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 15. dubna 2019

.....

Rada by som poďakovala Richardovi Římanovi a Ing. Zděňkovi Solnickému za možnosť absolvovania individuálnej odbornej praxe v ich firme a za odborné teoretické a praktické skúsenosti, ktoré som vďaka ním nadobudla. Ďalej by som chcela poďakovať Ing. Lenke Skanderovej, Ph.D., za konzultácie ohľadom bakalárskej praxe.

Abstrakt

V bakalárskej práci opisujem individuálnu odbornú prax vo firme Projektově.CZ s.r.o.. Práca zahŕňa informácie o firme a opis použitých technológií. Hlavnú časť práce predstavuje podrobný opis úloh, na ktorých som počas praxe pracovala. V závere hodnotím výhody a nové skúsenosti z odbornej bakalárskej praxe.

Kľúčové slová: React, JavaScript, HTML, CSS

Abstract

In the bachelor theses is described individual professional experience in company named Projektově.CZ s.r.o.. Theses includes information about company and description of used technologies. Main part of thesis represents detailed descriptions of job, which I have been worked on during my working experience. In the end of thesis, I evaluate advantages and new skills achieved from professional bachelor experience in company Projektově.CZ.

Key Words: React, JavaScript, HTML, CSS

Obsah

Zoznam použitých skratiek a symbolov

JS	–	JavaScript
DOM	–	Document Object Model
HTML	–	Hyper Text Markup Language
CSS	–	Cascading Style Sheets
URL	–	Uniform Resource Locator
UI	–	User interface

Zoznam obrázkov

Zoznam výpisov zdrojového kódu

1 Úvod

Na bakalársku prácu som si zvolila absolvovanie individuálnej odbornej praxe, ktorú som vykonávala vo firme Projektově.CZ. Hlavným dôvodom tohto výberu bolo, že som vo firme v pozícií vývojár nikdy nepracovala a nevedela som si predstaviť ako viacerí vývojári pracujú spoločne na jednom systéme. Ďalším dôvodom bolo získať odbornú prax a nové poznatky.

Podrobnejší opis spoločnosti Projektově.CZ, ich systému a moje pracovné zaradenie opisujem v prvej časti bakalárskej práce. V druhej časti sa nachádza popis technológií, ktoré som využívala počas odbornej praxe. Tretia časť obsahuje všetky úlohy na ktorých som pracovala. Pri každej úlohe opisujem dôvod vzniku úlohy, postup riešenia úlohy a čas, ktorý som strávila na vývoji úlohy. Na záver zhodnotím aké poznatky zo školy som využila pri vývoji úloh a aké nové poznatky a skúsenosti som nadobudla počas odbornej praxe.

2 Opis spoločnosti a pracovné zaradenie

2.1 O spoločnosti

Ostravská spoločnosť Projektově.CZ s.r.o bola založená 11.02.2013. Od založenia až po súčasnosť spolupracuje s univerzitou Vysoká škola báňská v Ostrave a s ďalšími odborníkmi z akademickej pôdy alebo partnerských firiem, ktorí sa zameriavajú na projektové riadenie [1].

2.2 Popis služby

Hlavným cieľom spoločnosti je poskytovať software ako službu dodávanú cez internet, skratkou SAAS (Software As A Service). Software umožňuje prehľadné riadenie firmy, projektov a úloh pomocou projektového riadenia. Jednoduché a prehľadné rozhranie umožňuje klientom vytvárať projekty a úlohy, sledovať ich priebeh a vývoj, a získať prehľad o konkrétnom zamestnancovi alebo celom tíme [1].

Spoločnosť poskytuje svoje služby na slovenskom a českom trhu v českom, slovenskom, nemeckom a anglickom jazyku. Projektově.CZ sa zameriava na malé a stredné firmy. Firmy môžu využívať software na akomkoľvek zariadení pripojenom na internet pomocou ľubovoľného internetového prehliadača. Kompletnú údržbu, aktualizácie a prevádzku systému poskytuje spoločnosť dodávajúca SAAS. [1]

2.3 Pracovné zaradenie

Do firmy Projektově.CZ som sa prihlásila na odbornú bakalársku prax na pozíciu frontend vývojára v React.js. Pred prijatím do firmy som absolvovala prijímací pohovor. Na tomto pohovore som mala ukázať projekt zo školy a podrobne opísať kód tohto projektu. Zároveň som mala doplniť kód v JavaScripte. Na konci pohovoru som odpovedala na otázky ohľadom mojich skúseností v programovaní.

Po prijatí som si mala naštudovať technológie, s ktorými som na praxi mala pracovať. Na začiatku praxe bolo potrebné nainštalovať do môjho notebooku všetky programy, ktoré boli nevyhnutné pre prácu so službou Projektově.CZ.

Na praxi som pracovala na viacerých úlohách. Pri úlohách mi pomáhal seniorský programátor. Prvé menšie úlohy boli určené na zorientovanie sa v systéme a pracovanie v Reacte. Hlavnou úlohou bolo navrhnúť a vytvoriť interaktívnu komponentu na filtrovanie projektov.

3 Technológie

Počas bakalárskej praxe som používala veľa známych technológií akými sú React a jeho komponenty, JSX alebo Flux.

3.1 React

Rect, tiež známy ako React.js alebo ReactJS, je známa JavaScriptová knižnica vyvíjaná spoločnosťou Facebook. Knižnica umožňuje vytvárať interaktívne používateľské rozhrania pre webové a mobilné aplikácie [2].

Výhodou reactu je virtuálny DOM, ktorý ukladá zmenené dáta a synchronizuje ich na reálny DOM. To umožní rýchlejšie obnovenie zmenených častí aplikácie, bez nutnosti načítania celej stránky. Pomocou Reactu sa môžu vytvárať opakovateľne používateľné komponenty [2].

3.2 JSX

JavaScript extension, alebo bežnejšie JSX predstavuje rozšírenie Reactu. Pomocou JSX môžeme písať stručné HTML štruktúry do rovnakého súboru ako JavaScriptový kód. V JSX je možné zjednodušiť vytváranie elementov React tým, že JSX využíva HTML syntax, bez volania JavaScriptovej funkcie React.createElement(). Takýmto zápisom sa zjednoduší a zmenší kód. Porovnanie kódov sa nachádza vo výpisoch 1 a 2 [2, 3].

```
return <div>Hello {this.props.name}</div>
```

Výpis 1: Ukážka JSX kódu

```
return React.createElement('div',null,'Hello ',this.props.name)
```

Výpis 2: Ukážka bez JSX kódu

JSX kód sa transformuje naspať na kód JavaScript, ktorý používa React.createElement(). Ak JSX kód nie je transformovaný, spôsobí to chybu jazyka JavaScript. Na takúto transformáciu JSX kódu sa používa nástroj *Babel* [2, 3].

Vďaka JSX môžeme v HTML kóde používať JavaScriptové objekty ohraničené zátvorkami. JSX môžeme použiť ako návratovú hodnotu alebo prijímací parameter funkcie [2, 3].

3.3 Komponenty

Komponenty v React by som prirovnala ku stavebným tehlám, ktoré sú základom pre vybudovanie stavby. Tak isto aj komponenty v Reacte sú samostatné časti, ktoré vhodným prepojením vytvárajú väčšiu a zmysluplnejšiu aplikáciu. Komponenta je funkcia alebo trieda JavaScriptu, ktorá prijíma vstupy a vracia elementy React, ktoré predstavujú časť užívateľského rozhrania.

Každá komponenta môže obsahovať ďalšie komponenty a tým vytvoriť strom komponent. Všetky komponenty sú znovupoužívateľné kdekoľvek v kóde [2, 3, 4].

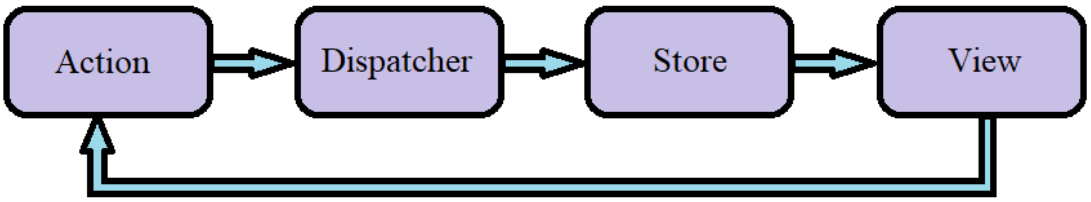
Každá z komponent obsahuje metódy, ktoré predstavujú životný cyklus komponenty. Tieto metódy sa vykonávajú v určitom poradí. Ako prvá metóda sa stále zavolá *constructor*, ktorá prijíma *props*. *Props* je objekt, ktorý obsahuje dáta z rodičovskej komponenty. Tieto dáta sú určené len na čítanie, preto by sme ich nemali nijako meniť. Vlastné dáta, ktoré môžeme rôzne modifikovať, sa v komponente nachádzajú v objekte *state* [2, 3, 4].

Ďalšou metódou životného cyklu je metóda *render*. Je to jediná povinná metóda v komponente. Volá sa automaticky pri každej zmene dát v komponente. Nemení objekty *state* a vracia *React element*, prípadne *null* alebo *false* [2, 3, 4].

Metóda *componentWillMount* sa volá pred pridaním do virtuálneho DOM a metóda *componentDidMount* po pridaní. *ComponentWillUnmount* sa zavolá po odobraní z virtuálneho DOM [2, 3, 4].

3.4 Flux

Flux je architektonický návrhový vzor, ktorý predstavuje jednosmerný tok dát aplikácie. Je navrhnutý tímom Facebooku. Flux sa skladá z troch hlavných častí: *Dispatcher*, *Store*, *View*. Pomocné metódy Dispathera sa nazývajú Action. O Action môžeme uvažovať ako o štvrtej časti Flux. Jednosmerný tok dát je znázornený na obrázku 1 [5].



Obr. 1: Jednosmerný tok dát

4 Úlohy

Počas bakalárskej praxe som pracovala ako frontend vývojar na viacerých úlohach. Pri úlohách som sa naučila pracovať s verzovacím systémom Git a na každú úlohu som vytvorila novú vetvu. Všetky úlohy boli programované v knižnici React.

4.1 Týždenný kalendár

Mojou prvou úlohou bolo vytvoriť týždenný kalendár úloh. Cieľom tejto úlohy bolo vytvoriť prehľadné užívateľské rozhranie na vykreslenie úloh v konkrétnom týždni. Celé UI reprezentujú komponenty *WeekCalendar* a *Week*. Keďže som nikdy predtým v Reacte nepracovala, bolo pre mňa ťažké takéto komponenty vytvoriť. Postupovala som podľa pokynov skúseného programátora, ktorý mi na tejto úlohe vysvetľoval ako sa v Reacte pracuje.

Ako prvé som sa naučila pracovať s knižnicou *Moment.js*. Potrebovala som ju na zistenie predchádzajúceho, aktuálneho a nasledujúceho týždňa. Jednotlivé týždne sa dali posúvať pomocou tlačidiel *Nasledujúci* a *Predchádzajúci*. Aby sa mohol užívateľ lepšie zorientovať, nad každým vykresleným týždňom sa nachádza číslo daného týždňa, ktoré som tiež získala z *Moment.js*. Na vykreslenie úloh v týždni bolo potrebné získať dáta zo *Store*, konkrétne *TaskStore* pomocou funkcie *getAll*.

Každá úloha obsahovala vlastné ID, názov, názov projektu, ID riešiteľov, dátum začiatku a konca. Podľa dátumu začiatku úlohy som odfiltrovala iba tie úlohy, ktoré som potrebovala na konkrétny týždeň. Filtrovanie úloch sa nachádza vo výpise 3.

```
thisWeek = thisWeek.filter(t => t.dueDate && t.dueDate >= startOfTheWeek && t.
    dueDate <= endOfTheWeek)
```

Výpis 3: Filtrovanie úloh

Keďže všetky vyfiltrované úlohy patrili celému tímu, do ktorého užívateľ patril, bolo potrebné rozdeliť úlohy podľa členov tímu. Vďaka tomu vznikol filter, ktorý umožňoval vyfiltrovať vlastné úlohy ale aj úlohy konkrétnych kolegov alebo celého tímu. Na to aby sa filter vytvoril som potrebovala získať ID, meno a priezvisko všetkých členov tímu. Všetky dáta sa nachádzali v *UserStore*. Na určenie práve prihláseného užívateľa som využila funkciu *getCurrent*.

Každá úloha sa vykreslila v týždňovom kalendári, ak mala v daný dátum termín konca. Preto vznikol ďalší filter, v ktorom môžeme zvoliť zobrazenie úloh v celom ich trvaní alebo len s končiacim termínom. Nie všetky úlohy mali začiatok a koniec v rovnakom týždni. Preto bolo potrebné tieto úlohy vyfiltrovať tak, aby zobrazovali svoj priebeh len v konkrétnom týždni.

Na tejto úlohe som sa naučila ukladať nastavené filtre do URL. Vďaka adrese URL si používatelia budú môcť nastaviť rovnaké filtre v týždennom kalendári.

Aplikáciu môžeme využívať v štyroch rôznych jazykoch, preto bolo potrebné celý týždenný kalendár preložiť. Na preklad slúži knižnica *i18n.js*. Vo výpise 4 je použitá táto knižnica.

```
<span> {numberOfWeek} {I18n.t('issues.week_calendar.week_name')} </span>
```

Výpis 4: Ukážka použitia knižnice *i18n.js*

Na vykreslenie užívateľského rozhrania som potrebovala použiť HTML a CSS. Týždenný kalendár som potrebovala rozdeliť na sedem rovnakých častí, preto som zvolila tag *<table>*. Každá časť predstavovala jeden deň v týždni, v ktorom sa nachádzal dátum, názvy úloh a ich projektov. Aktuálny deň v týždni sa na odlíšenie od ostatných dní zvýraznil inou farbou. Preto som prvýkrát použila classnames knižnicu, ktorá predstavuje jednoduchý nástroj pre podmienené spojenie viacerých className dohromady. Podmienené štýly sú znázornené vo výpisoch 5 a 6.

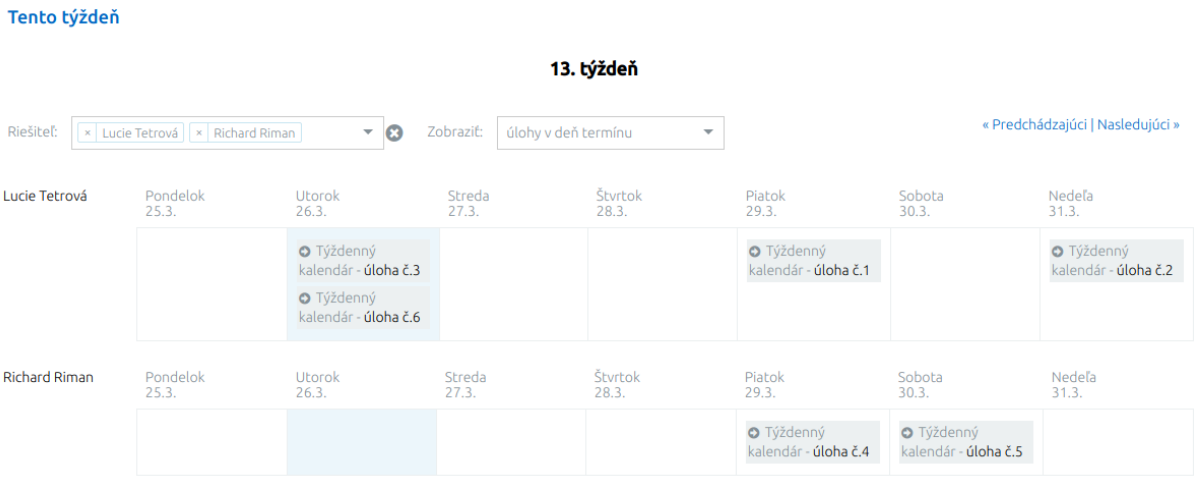
```
&__dayContent {
color: $color-black;
background-color: $color-white;
border: 1px $color-light-bg solid;

  &--today {
  background-color: mix($color-white, $color-info-background, 50);
  }
}
```

Výpis 5: Podmienené štýly v CSS kóde

```
weekTasks.push( <td className={cn('WeekCalendar__dayContent',
              { 'WeekCalendar__dayContent--today': day.startOf('day')
              .isSame(moment().startOf('day')) })} key={i}>{rows}</td>)
```

Výpis 6: Podmienené štýly v JSX kóde



Obr. 2: Týždenný kalendár s filtrom

Výsledný týždenný kalendár je zobrazený na obrázku 2. Na tejto úlohe som sa naučila veľa nových vecí, ktoré som využila aj pri ostatných úlohách. Keďže to boli moje prvé skúsenosti v React, musela som nad touto úlohou stráviť viac času. Celkovo som týždenný kalendár programovala osem dní.

4.2 Flat avatar

Meno a priezvisko užívateľa zahŕňa veľa znakov. V prípade, ak potrebujeme vypísať všetkých členov tímu potrebujeme veľmi veľa miesta na ich zobrazenie. Na takéto prípady Projektově.CZ využíva flat avatary. Flat avatar predstavuje iniciály mena a priezviska v kruhu. Pri vytvorení každého užívateľského účtu užívateľovi systém vygeneruje flat avatara s náhodnou farbou. Nie každý užívateľ bol spokojný s pridelenou farbou. V niektorých firmách mali viacerí kolegovia pridelenú rovnakú farbu. Mojou úlohou bolo, vylepšiť flat avatara tak, aby si každý užívateľ mohol nastaviť svojho flat avatara podľa vlastného výberu farby.

Na túto úlohu som vytvorila komponentu *MyColor*. Farby som uložila do objektu. Každú farbu predstavuje hexadecimálne číslo. Ukážka farieb v objekte *color* sa nachádza vo výpise 7.

```
color = {1:'#8adbcb', 2:'#94e3b5', 3:'#97c9ea', 4:'#caa9d8', 5:'#97a1ac',
        6:'#88cdbf', 7:'#90d4ad', 8:'#91bdd9', 9:'#c49fd3', 10:'#f5df84',
        11:'#f0bc8e', 12:'#f0a39b',13:'#f6cb86', 14:'#e6a77d', 15:'#dd9992',
        16:'#dbdee0', 17:'#bcc3c3', 18:'#97a1ac'}
```

Výpis 7: Farby v objekte *color*

V UI sa každá farba vykresľuje do malého obdĺžnika. Každému farebnému obdĺžniku sa zvýrazní okraj po prejdení myšou. Po kliknutí na obdĺžnik s danou farbou sa v strede obdlžníka vykreslí ikona *icon-p-done*.

Zmena farby flat avatara sa nachádza na mieste *Moje nastavenia*. Farby sa zobrazia až po kliknutí na *Zmeniť farbu*. Na obrázku 3 sú zobrazené všetky farby, z ktorých si užívateľ môže vybrať. Aby si užívateľ mohol svojho flat avatara zmeniť na vybranú farbu, musel svoje rozhodnutie potvrdiť tlačidlom *uložiť*.

Obr. 3: Zmena farby flat avatara

Nie všetci užívatelia používajú flat avatara, pretože v systéme je možné použiť Google avatar. Takýmto užívateľom sa nezobrazuje možnosť *Zmeniť farbu*. Neskôr, k zmene farby pridala firma novú funkcionalitu *Nahrať vlastný obrázok*.

Tejto úlohe som sa venovala štyri dni. Moje nastavenia neboli naprogramované v React, preto najťažším na tejto úlohe bolo pridať moju React komponentu do existujúcého kódu a tiež k tomu prispôsobiť vzhľad.

4.3 Pridávanie časových záznamov

Užívatelia si ku každej úlohe, na ktorej pracujú, môžu pridávať strávený čas. Niektoré firmy podľa strávených časov vypočítavajú zamestnancom výplaty. Aby si užívateľ zadal čas na konkrétnej úlohe, musel si ju nájsť medzi všetkými úlohami a až v jej detaile si mohol pridať čas. Na pridanie času sa užívateľovi zobrazí dialógové okno. Užívateľ okrem času musí zadať dátum a vybrať aktivitu. Nepovinným údajom je komentár k úlohe. Manažér firmy môže pridávať čas iným kolegom. Dialógové okno je programované v React.

Ďalším miestom, kde si užívatelia môžu pridať časový záznam je *Strávený čas*. Na tomto mieste sa nachádzajú všetky časové údaje nad ktorými užívateľ pracoval. Po kliknutí na tlačidlo pridať čas sa užívateľovi zobrazilo dialógové okno, kde si mohol vybrať úlohu a pridať čas. Takéto pridávanie času trvalo dlhšie ako pri pridávaní v detaile úlohy, pretože toto dialógové okno nebolo programované v React.

Mojou treťou úlohou bolo nahradiť dialógové okno v *Strávený čas* a rozšíriť ho na pridávanie času k ľubovoľným úlohám ale aj projektom. Nové rozšírene dialógové okno som mala pridať na nové miesto *Moja stránka* do bloku *Môj strávený čas* pomocou tlačidla plus, pozri obrázok 4.

Obr. 4: Tlačidlo plus na pridávanie stráveného času

Ako prvé som dialógové okno pridala do už spomínaných miest zavolaním komponenty *SpentTimeDialog*. Volanie metódy je znázornené vo výpise 8.

```
<SpentTimeDialog show currentUser={this.props.currentUser} onClose={this.
    onCloseTimeDialog}/>
```

Výpis 8: Zavolanie komponenty *SpentTimeDialog*

Na zobrazovanie projektov som potrebovala dáta, ktoré som získala z *ProjectStore*. Na vykreslenie projektov som použila komponentu *SelectField*, ktorá prijímala dáta len ako objekt obsahujúci štruktúru label a value. V metóde *getSelectProjects* som pomocou funkcie *map* vytvorila potrebnú štruktúru, viď výpis 9.

```
getSelectProjects() {
 return this.props.projects.map(t => ({ label: t.get('name'),
                                value: t.get('id') })).toArray()
}
```

Výpis 9: Metóda *getSelectProjects*

Pri vybraní konkrétneho projektu sa užívateľovi majú zobraziť len úlohy tohto projektu. Ak sa zmení projekt, musia sa zmeniť aj úlohy projektu. Pri každej zmene sa zavolá metóda *onChangeProject*, v ktorej nastavím vybraný projekt a pomocou funkcie *httpGet* získam zo servera dáta úloh. Ukážka získania dat sa nachádza vo výpise 10. Na zobrazenie úloh tiež použijem *SelectField*, preto v tejto metóde upravím dáta do vhodnej štruktúry.

```
onChangeProject = (option) => {
    this.setState({ project: option })
    httpGet(`/api/issues?scope=all&projectId=${option.value
            &onlySelf=1&attributes=id,subject,statusId,totalSpentHours,
            spentHours&limit=1000`)
            .then(({ issues }) => {
               this.setState({ tasks: issues.map(i => ({ value: i.id,
               label: i.subject, status: i.statusId,
               totalSpentHours: i.totalSpentHours, spentHours: i.spentHours}))
```

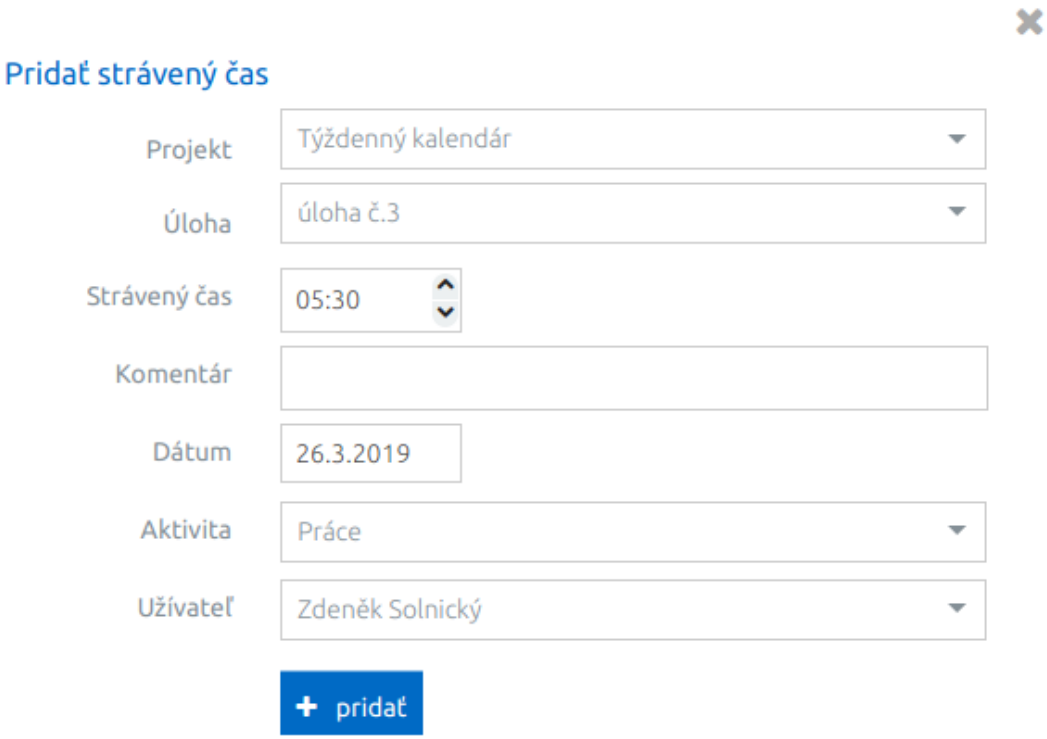
```
            })
        })
    }
```

Výpis 10: Získanie dát pomocou funkcie *httpGet*

Strávený čas projektov alebo úloh sa pridá, až keď užívateľ klikne na tlačidlo pridať. V tom momente sa zavolá metóda *onAddTimeEntry*, v ktorej pomocou funkcie *httpPost* zasielam dáta na server, pozri výpis 11. Po zaslaní sa zavolala metóda *onClose* z rodičovskej komponenty na uzatvorenie dialógového okna.

```
httpPost('/time_entries.json', { timeEntry: { hours,
      spentOn: date.format('YYYY-MM-DD'),comments: comment,
      activity_id: activityType, userId, issueId, projectId } })
```

Výpis 11: Zasielanie dát pomocou funkcie *httpPost*



Obr. 5: Dialógové okno na Strávený čas

Diaógové okno strávených časov je zobrazené na obrázku 5. Na tejto úlohe som sa naučila získavať dáta zo serveru a tiež ich posielať na server. Celkovo som nad úlohou strávila štyri dni.

4.4 Úvodný sprievodca

Pri prvom prihlásení do systému môžu byť niektorí užívatelia z rozsiahleho systému zmätení a nevedia, čo ako prvé vykonať. Obzvlášť administrátori, ktorí riadia celý tím. Administrátor by mal ako prvé vytvoriť projekty, prípadne podprojekty a vytvoriť nových členov tímu.

Mojou úlohou bolo vytvoriť úvodného sprievodcu pre administrátorov. Cieľom tejto úlohy je pomôcť administrátorovi vytvoriť prvé projekty a pridať k ním nových členov. Ďalšími požiadavkami úlohy bolo, aby sa úvodný sprievodca zobrazil len pri prvom prihlásení, pomocou dialógového okna, vo viacerých krokoch. Dialógové okno má byť povinné a nemá byť možné ho preskočiť alebo zrušiť.

Na začiatok som spolu s kolegami prebrala, čo sa bude v jednotlivých krokoch nachádzať a navrhli sme UI, ktoré sme počas implementácie vylepšovali. Ďalej som vytvorila komponentu *WelcomeWizard*, v ktorej som programovala všetky navrhnuté kroky. Na určovanie aktuálneho kroku som použila stavovú premennú *window*. V každom vykreslenom okne sa nachádzajú malé farebné krúžky s číslom, ktoré reprezentujú kroky sprievodcu. Farby odlišujú predchádzajúce, aktuálne a nasledujúce kroky. Pod každým krúžkom je text opisujúci daný krok. Krúžky s textom reprezentuje komponenta *WizardPosition*.

Prvý krok nebol na programovanie náročný, pretože okrem farebných krúžkov sa v ňom nachádzal uvítací text a tlačidlo na ďalší krok. V druhom kroku sa vytvárajú projekty a podprojekty. Počas návrhu UI padlo viacero návrhov, ako riešiť tento krok tak, aby to bolo pre užívateľa intuitívne a čo najjednoduchšie. Nakoniec sme sa zhodli vytvoriť predpripravenú šablónu s pomocnými tlačidlami a pridať obrázky s názornými ukážkami, ktoré je možné vidieť na obrázku 6. Obrázky som najprv musela pridať do súboru *images* a pomocou tagu *<img>* načítať obrázok podľa zvolenej cesty v *src*, viď výpis 12.

```
<img className="Images__thumbnails" src="/images/welcome-subprojects.png" />
```

Výpis 12: Načítanie obrázku

Obr. 6: Pridávanie projektov

Každý projekt môže mať podprojekt, ktorý môže mať svoj vlastný podprojekt, takzvaný „podpodprojekt“. Podprojekty sú v sprievodcovi odsadené doprava, aby užívateľ rozlíšil, čo je projekt a čo je jeho podprojekt. Pomocou šípok vpravo sa dajú vytvoriť podprojekty. Každý podprojekt musí mať hlavný projekt a „podpodprojekt“ tiež musí mať podprojekt. Preto bolo nutné ošetriť situácie s posunutím šípky vpravo, aby nevznikol podprojekt bez hlavného projektu a „podpodprojekt“ bez podprojektu, alebo posunutie o viacero úrovní, pozri výpis 13.

```
{p.depth <=1 && idx !== 0 && !(projects[idx].depth == 2 || idx !== 0
          && (projects[idx - 1].depth + 1 == projects[idx].depth)) &&
<i className="icon-chevron-right Welcome__iconRight"
   onClick={this.arrowRight.bind(this, idx)}
/>}
```

Výpis 13: Vykreslenie šípky po splnení podmienok

Šípkou vľavo sa z podprojektov vytvoria naspať projekty s posunutím o jednu úroveň. Tak ako šípka vpravo, aj šípka vľavo sa vykreslí iba vo vhodných situáciách. Po stlačení šípky sa zavolá metóda *arrowRight(idx)* alebo *arrowLeft(idx)*, v ktorej sa projekt s indexom *idx* nastaví na zvolenú úroveň.

Okrem šípok je pomocným tlačidlom krížik, ktorý umožní zmazať vhodný projekt alebo podprojekt. Nemôže sa zmazať projekt, za ktorým nasleduje projekt o úroveň menší.

Užívateľ je povinný zadať aspoň jeden projekt, preto posledný projekt nie je možné zrušiť. Stlačením tlačidla sa zavolá metóda *deleteProject*, v ktorej sa zisťuje, či je možné vymazať projekt, a ak áno, posunie sa poradie projektov Lodash funkciou *compact*. Metóda *deleteProject* je znázornená vo výpise 14.

```
deleteProject(project) {
let projects = [...this.state.projects]
    if(projects.length == 0 || projects.length == 1) return
    if ( projects[project + 1] && projects[project].depth < projects[project +
        1].depth ) return
    else {
      this.setState({focused: project})
      delete projects[project]
      projects = _.compact(projects)
    }
    this.setState({ projects })
  }
```

Výpis 14: Metóda *deleteProject*

V prípade, ak užívateľ potrebuje vytvoriť viac projektov, ako je vo vytvorenej šablóne, alebo viacero projektov z nej vymazal, a rozhodol sa ich pridať naspať, môže vytvoriť ďalšie projekty stlačením tlačidla Enter. Aby užívateľ nemohol pridávať projekty v úvodnom sprievodcovi donekonečna, dohodli sme sa, že maximálny počet projektov a podprojektov bude osem.

Pridanie projektov na server bolo veľmi náročné naprogramovať, aj keď pre užívateľa to bolo iba stlačenie tlačidla *pridať projekty*. Týmto tlačidlom sa postupne volajú metódy na vytvorenie projektov a podprojektov . Ako prvé sa zavolala metóda *onStep3*. V nej sa na začiatok nastavuje premenná *savingProjects* na *true*, v dôsledku čoho prebieha pridávanie projektov na server a tlačidlo *pridať projekty* je vtedy neaktívne a zmení farbu. Je to ochranné opatrenie pred viacerými stlačeniami tlačidla užívateľom, aby sa nepridali tie isté projekty viackrát. Aby užívateľ videl, že sa na ukladaní pracuje, do tlačidla sa pridala ikona *spinner*.

Ďalej sa pomocou podmienok zisťuje, či vyplnený podprojekt má aj vyplnený názov hlavného projektu alebo sa tam nachádza aspoň jeden projekt. Ak nie, užívateľovi sa objaví hláška, prečo sa projekty nedajú pridať a premenná *savingProjects* sa nastaví na false. Ak áno, zavolá sa asynchrónna metóda *createProjects*. V nej sa postupne pomocou cyklu *for* prechádzajú projekty. Každý projekt sa posiela do asynchrónnej metódy *createProject*, ktorá prijíma parametre *name* a *parentId*. Metóda čaká, dokedy sa projekt s danými parametrami uloží na server a následne vráti uložený projekt s priradeným id späť do metódy *createProjects*, pozri výpis 15.

```
async createProject(name, parentId) {
    let response = await httpPost('/projects.json', { project: { name, parentId
        } })
    ProjectActions.ensureProjects({ force: true })

    return response.project
  }
```

Výpis 15: Asynchrónna metóda *createProject*

Metóda *createProjects* čaká, až kým sa uložený projekt nevráti. Z vráteného projektu sa uloží pridelené id projektu, ktoré je potrebné ako parenId pri podprojektoch. Všetky vytvorené projekty sa ukladajú do poľa. Po pridaní všetkých projektov na server sa pole projektov pošle do metódy *onStep3*, viď výpis 16.

```
async createProjects() {
    let projects = []

    for (let i = 0; i < this.state.projects.length; i++) {
      const p = this.state.projects[i]
      let parentId = null
      if (p.name && p.name.length !== 0) {
        if (p.depth > 0) {
          const idx = _.findLastIndex(projects, { depth: p.depth - 1 })
          parentId = projects[idx].id
        }
        let project = await this.createProject(p.name, parentId)
        p.id = project.id

        project.depth = p.depth
        projects.push(project)
      }
    }
    return projects
  }
```

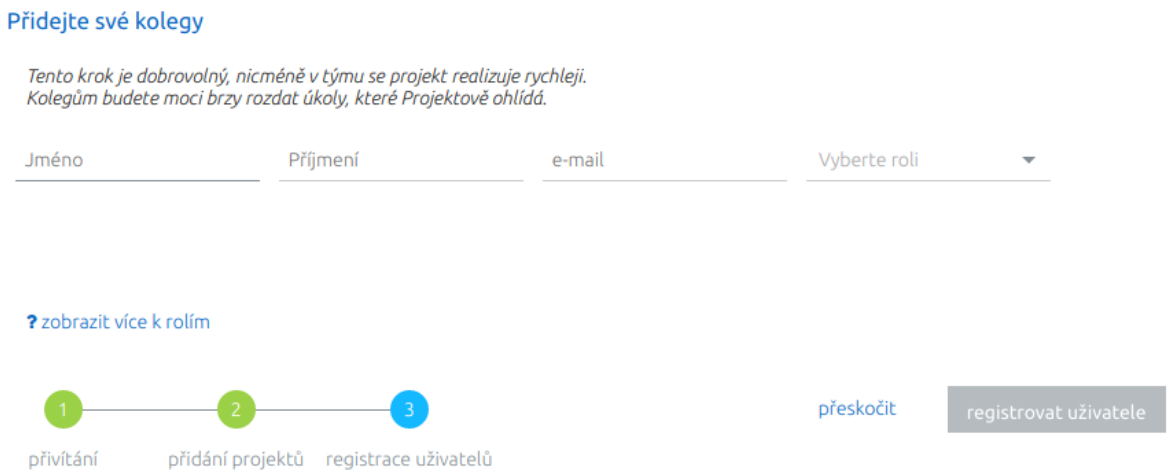
Výpis 16: Asynchrónna metóda *createProjects*

Po dokončení onStep3 sa vykreslí UI ďalšieho kroku sprievodcu. Tretím krokom je pridávanie nových členov. Na rozdiel od pridávania projektov je pridávanie užívateľov nepovinné a užívateľ ho môže preskočiť.

Nového kolegu užívateľ pridá tak, že vyplní jeho meno, priezvisko, e-mail a rolu. Rolu užívateľa môže vybrať z viacerých možností. Popis možností sa zobrazí kliknutím na nápovedu.

V metóde *onAddUserIfLastNotEmpty* sa zistí, či sú všetky údaje vyplnené. Na testovanie správne zadaného e-mailu som použila naprogramovanú metódu *validEmail*. Ak je všetko vyplnené, zobrazí sa nový riadok na pridanie ďalšieho užívateľa. V úvodnom sprievodcovi môže administrátor pridať maximálne 5 členov.

Pridávanie kolegov prebieha podobne ako pridávanie projektov. Stlačením tlačidla registrovať užívateľov sa zavolá metóda *onStep4* a tlačidlo bude neaktívne. V metóde sa opäť zisťujú správne vyplnené údaje. Pri nesprávnom vyplnení sa zobrazí hláška o tom, čo je nesprávne a tlačidlo sa nastaví na aktívne. Ak sú všetky údaje správne, zavolá sa asynchrónna metóda *createUsers*. Pri každom užívateľovi sa zavolá metóda *createUser*, ktorá prijíma meno, priezvisko a e-mail, ktoré následne uloží na server a pridá nového člena tímu do systému. Po pridaní užívateľa sa zavolá metóda na pridávanie rolí. Rola užívateľa sa pridáva ku konkrétnemu projektu. Pridávanie prebieha v asynchrónnej metóde *createRoles*, odkiaľ je volaná asynchrónna metóda *createRole*, ktorá prijíma id užívateľa, id projektu a id roly, ktoré následne pridá na server. Asynchrónne metódy prebiehajú rovnako ako metódy na pridávanie projektov. Na obrázku 7 sa nachádza pridávanie užívateľov.



Obr. 7: Pridávanie užívateľov

Po uložení nových užívateľov sa vykreslí ďalšie okno, v ktorom administrátor môže vstúpiť do projektov v systéme pomocou tlačidla *prejsť na projekty* a tým ukončiť úvodného sprievodcu. Okrem toho sa užívateľovi navrhne možnosť vytvoriť šablóny projektov. Vytváranie šablón pokračuje v ďalšom kroku úvodného sprievodcu.

Na pridávanie šablón som vytvorila novú komponentu *TemplatesWizard*. Šablóna slúži ako nečisté naplánovanie projektov, do ktorej sa môžu pridávať úlohy aj členovia tímu. Z pripravenej šablóny sa dá vytvoriť projekt so zvolenými úlohami a členmi tímu.

Šablóny, tak ako projekty, existujú v troch úrovniach, preto na vytváranie šablón využívam rovnaký princíp ako pri pridávaní projektov. Užívateľ tak ako v projektoch môže vytvárať úrovne šablón pomocnými tlačidlami. V tomto sprievodcovi môže pridať maximálne tri šablóny. Asynchrónnymi metódami *createTemplates* a *createTemplate* pridá šablóny na server. Ak administrátor vytvoril nových užívateľov, v asynchrónnych metódach *createRoles* a *createRole* ich pridá k šablónam.

Po pridaní šablón sa objaví posledné okno, v ktorom sa nachádza informácia ako rozoznať v systéme šablóny medzi projektami a tip na využitie myšlienkovej mapy. Z tohto okna užívateľ prejde rovno do projektov.

Táto úloha bola oveľa náročnejšia ako predchádzajúce, nie len na implementáciu ale aj návrh UI. Novinkou pre mňa boli asynchrónne metódy a princíp akým fungujú. Na tejto úlohe som pracovala dvanásť dní.

4.5 Filtre projektov

Hlavnými časťami v službe Projektově sú projekty a ich úlohy. Po stlačení tlačidla projekty sa zobrazí zoznam všetkých vytvorených projektov. Okrem názvu projektu sa zobrazí aj dátum plánovaného začiatku, dátum plánovaného termínu, avatary členov tímu a vlastné polia.

Vyhľadanie konkrétneho projektu medzi stovkami, či tisíckami projektov bolo veľmi náročné. V systéme sa donedávna dali vyhľadať projekty iba po zadaní textu. Užívateľ si nemohol zobraziť projekty konkrétneho kolegu alebo viacerých kolegov. Problémom bolo aj vyhľadanie projektov s blížiacim sa termínom alebo plánovaným začiatkom. Každá firma má nadefinované vlastné polia, v ktorých zadáva potrebné údaje o projekte. Konkrétny údaj z vlastného poľa bolo tiež zložité nájsť medzi všetkými projektami.

Na vyriešenie týchto problémov bolo nutné vytvoriť filtre projektov. Mojou úlohou bolo vytvoriť filtre na filtrovanie podľa názvu, skratky, plánovaného začiatku, plánovaného termínu, členov tímu a vlastných polí projektov a vytvoriť vhodnú kombináciu týchto filtrov.

Začala som ako zvyčajne tým, že som si vytvorila novú komponentu *Filter*, ktorá sa volá z rodičovskej komponenty *Projects*. Ako prvé som v komponente začala riešiť vyhľadávanie projektov podľa názvu a skratky. V komponente som si deklarovala stavovú premennú *text*, do ktorej sa ukladá zadaný text užívateľom. Text sa zadáva pomocou *Autosizeinput* a pri každej zmene sa zavolá metóda *onChageText*, v ktorej sa zmení stav premennej *text* na aktuálne zadaný text. Text, ktorý užívateľ zadal, je jedno z kritérií, podľa ktorého sa majú vyfiltrovať projekty. Na všetky kritéria používam metódu *generateCriteria* a objekt *criteria*. Text ukladám do objektu *criteria.name*. Všetky kritéria posielam do rodičovskej komponenty *Projects*, konkrétne do metódy *onChangeFilter*, ktorá ich zasiela do *ProjectStore*. V *ProjectStore* som upravila metódu *getVisibleProjects* tak, aby vyfiltrovala projekty podľa poslaných kritérií. Vo výpise 17 je zná-

zornené fitrovanie projektov podľa kritéria *name*.

```
if (filter.criteria.name){
          filtered = filtered.filter(p => (_.includes(downcode(p.name).
             toLowerCase(), downcode(filter.criteria.name).toLowerCase())) || (
             p.shortcut != null &&
           _.includes(downcode(p.shortcut).toLowerCase(), downcode(filter.
              criteria.name).toLowerCase()))
}
```

Výpis 17: Filtrovanie projektov v *ProjectStore* podľa názvu a skratky projektu

Na zobrazenie ďalších filtrov som vytvorila komponentu *FilterDropDownMenu* a kaskádové štýly som využila z inej časti programu. Menu s filtrami sa užívateľovi objaví po kliknutí do *Autosizeinput*. Konkrétny filter si vyberie kliknutím na jeho názov, a tým sa zobrazí ďalšie menu s konkrétnymi hodnotami filtra. Hodnoty sa uložia do objektu *selectItems* a pošlú sa do komponenty *Select*, ktorá ich v menu zobrazí.

Vybratím filtru *Plánovaný začiatok* a *Plánovaný termín* sa zobrazia hodnoty *Dnes*, *Tento týždeň*, *Tento mesiac*, *Kalendár* a *Nezadané*. Pri kliknutí na možnosť *Kalendár* sa užívateľovi objaví dialógové okno kalendára, v ktorom si môže vybrať časové rozmedzie s konkrétnymi dátumami. Hodnotami filtra *Členovia tímu* sú všetci členovia tímu. Vybraná hodnota sa pošle späť do komponenty *FilterDropDownMenu* a v metóde *onChangeSelect* sa zavolá metóda *addFilter* rodičovskej komponenty *Filter*, do ktorej sa pošle vybraný filter a hodnota filtra. V tejto metóde prechádzam všetky filtre, a v prípade, ak sa tam takýto filternenachádza, pridám ho do poľa objektov *filters*, viď výpis 18. Každý objekt *filters* obsahuje *type*, do ktorého sa ukladá typ filtra a *value*, do ktorej sa ukladajú hodnoty filtra.

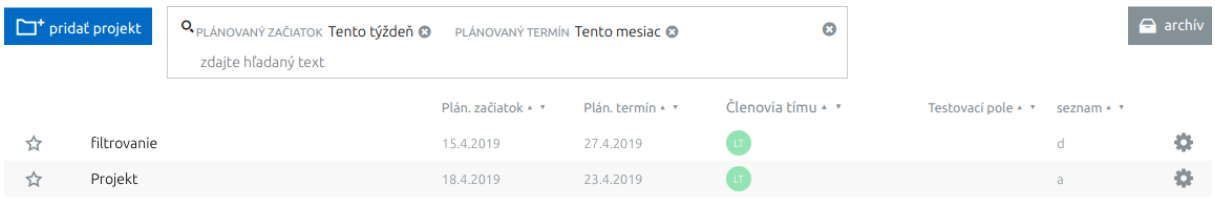
Z filtrov *Plánovaný začiatok* a *Plánovaný termín* si užívateľ môže zvoliť stále iba jednu hodnotu filtra. Ak do metódy zašlem filter *Plánovaný začiatok* alebo *Plánovaný termín* a zistím, že takýto filter sa tam už nachádza ale s inou hodnotou, nastavím tomuto filtru novú hodnotu. S filtrom *Členovia tímu* je to komplikovanejšie, pretože tento filter má zobraziť projekty viacerých členov tímu a nie len jedného. Ak užívateľ vyberie viac hodnôt tohto filtra, pridajú sa ako viaceré samostatné filtre s konkrétnou hodnotou, ktorá sa nemení na inú hodnotu tak, ako v prípade plánovaného začiatku a termínu.

```
addFilter = (name, value) => {
let filters = [...this.state.filters]
let isInFilter = false
if (_.includes(['plannedDueDate','plannedStartDate', 'member'], name)) {
   if (typeof value === 'object') {
     value = { start: value.start.valueOf(), end: value.end.valueOf()   }
```

```
    }
    if (!filters[0]){
      filters.push({type: name, value: value})
    } else if (filters[0].type == null) {
        filters[0].type = name
        filters[0].value = value
    } else {
     for (let i = 0; i < filters.length; i++) {
         if (filters[i].type == name && name != 'member') {
            filters[i].value=value
            isInFilter = true
         }
         if (filters[i].type == name && name == 'member' && filters[i].value ==
             value) {
           filters[i].value=value
           isInFilter = true
         }
     }
     if (isInFilter == false) {
           filters.push({type: name, value: value})
      }
this.setState({ filters}, () => {
    this.addFiltersToURL(filters)
    this.props.onChange(this.generateCriteria())
    })
}
```

Výpis 18: Metóda *addFilter*

Nový filter sa uloží do *state* a v metóde *generateCriteria* vytvorí nové kritéria filtrov, ktoré sa cez komponentu *Projects* presunú do *ProjectStoru*, kde sa vyfiltrujú projekty podľa poslaných kritérií. Vyfiltrované projekty podľa plánovaného začiatku a plánovaného termínu sú znázornené na obrázku 8.

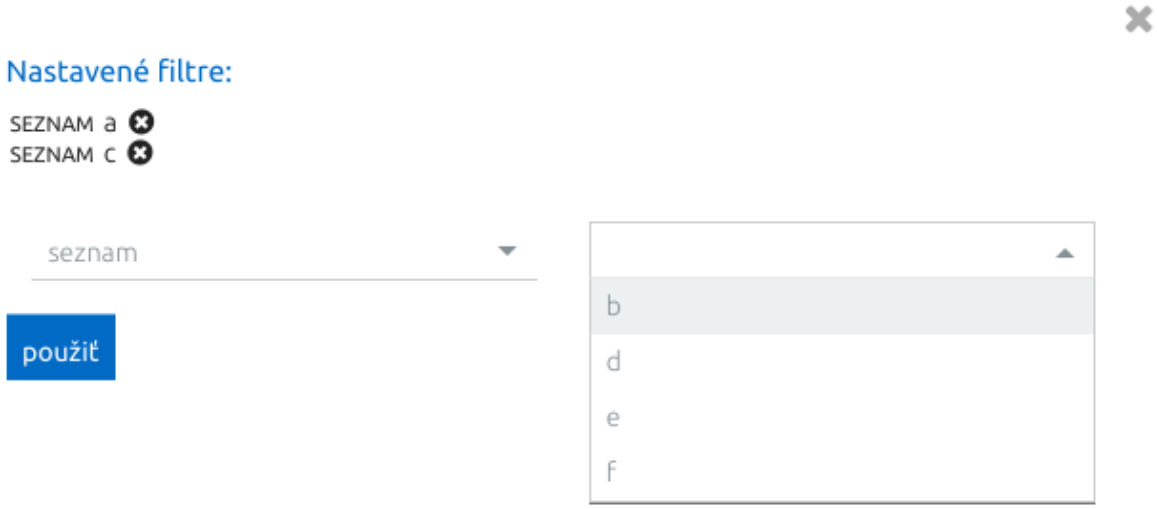


Obr. 8: Vyfiltrované projekty podľa plánovaného začiatku a plánovaného termínu

Filtre samostatných polí sú komplikované a preto ich riešim osobitne v komponente *CustomFieldDialog*, ktorá sa zavolá z komponenty *FilterDropdownMenu* po vybratí filtru *Samostatné pole*. Užívateľovi sa objaví dialógové okno, v ktorom si môže vybrať konkrétne samostatné pole a prideliť mu hodnotu, podľa ktorej chce filtrovať projekty. Dialógové okno vlastných polí je znázornené na obrázku 9. Samostatné polia sa zobrazujú v *InlineSelectField*.

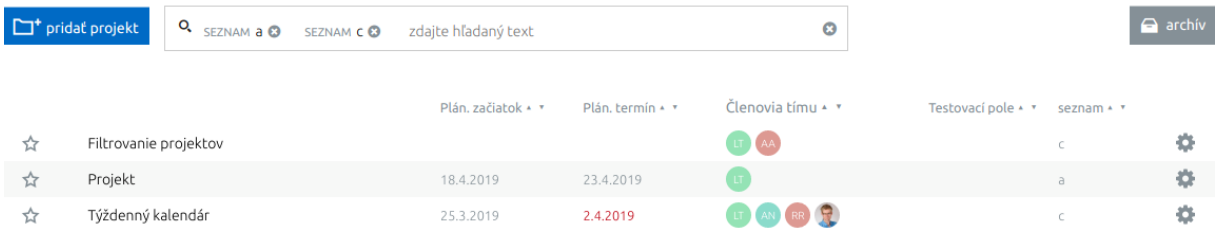
Hodnoty samostatného poľa môžu byť uložené ako *list*, *text*, *string* a *int*. Samostatné pole s typom hodnôt *list*, zobrazí hodnoty pomocou *InlineSelectField*. Iba v tomto type môže užívateľ zadať viacero možností a tým pridať viaceré filtre. Spôsob filtrovania podľa takýchto filtrov je pomocou „or“, takže sa zobrazia projekty spĺňajúce aspoň jeden z týchto filtrov. Ak samostatné pole má hodnoty typu *text* a *string*, zobrazí sa *InlineTextField*, do ktorého užívateľ zadá text. Podľa tohto textu sa neskôr vyfiltrujú projekty samostatného poľa, ktoré obsahujú hodnotu, v ktorej sa nachádza zadaný text.Podobným typom je aj *int*, do ktorého sa zadáva iba číslo. V týchto filtroch je spôsob filtrovania pomocou „and“.

Vybranú hodnotu posielam do metódy *findValueInCustomFields* alebo *findTextInCustomFields*, v ktorých vyhľadávam id projektov podľa vybraného vlastného poľa a hodnoty. Do objektu *customFieldsIdWithProjectsId* sa uloží id vlastného poľa, hodnota, id projektov a spôsob filtrovania. Všetky filtre vlastných polí sa zobrazia vo vrchnej časti dialogóveho okna. Pri každom filtri je tlačidlo na zrušenie filtra. Pri zrušení sa zavolá metóda *removeCustomFieldValue*.



Obr. 9: Dialogové okno filtrov vlatných polí

Pre zobrazenie projektov vyfiltrovaných podľa vlastných polí užívateľ stlačí tlačidlo *použiť*. Vtedy sa zavolá metóda *saveCustomFieldValues*, ktorá zašle dáta do komponenty *FilterDropdownMenu* a tá ich prepošle do komponenty *Filter* zavolaním metódy *addFiltersFromCustomFields*. V tejto metóde vytváram pole id projektov *projectsIdFilterByOr*, ktoré sa filtrujú podľa spôsobu „or“ a druhé pole *projectsIdFilterByAnd* podľa spôsobu „and“.

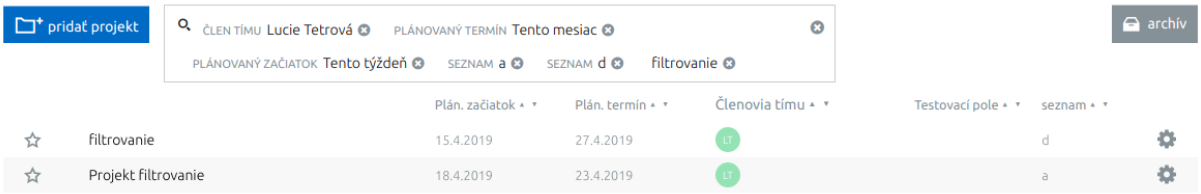


Obr. 10: Vyfiltrované projekty podľa vlastných polí

Všetky id projektov, ktoré sa majú zobraziť z filtrov vlastných polí vyberám v metóde *setProjectsIds*. Najprv pomocou Lodash funkcie *intersection* vytváram prvý prienik id projektov *projectsIdFilterByAnd* a druhý prienik z id projektov podľa spôsobu „or“ a prvého prieniku. Výsledný prienik obsahuje všetky potrebné id projektov a filter vlastných polí s týmito id vytvára ďalšie kritérium, ktoré sa zašle do *ProjectStoru*. Vyfiltrované projekty podľa vlastných polí sú zobrazené na obrázku 10.

Po opätovnom otvorení dialogóveho okna sa v ňom zobrazia všetky navolené filtre vlastných polí. Tieto filtre sa získajú v metóde *setCustomFieldsFromFilters* prostredníctvom *props* a uložia sa do *state*.

Všetky zvolené filtre sa vypíšu v *Autosizeinput* s konkrétnym názvom filtra a hodnotou. Na výpis zvolených filtrov používam komponentu *Tags*. Kaskádové štýly, ktoré vykresľujú filtre boli naprogramované v inej časti programu. Pri všetkých hodnotách filtrov sa nachádza krížik, ktorým sa odoberie konkrétny filter. Filtre vlastných polí sa odoberú v metóde *deleteCustomFieldsFilter* a ostatné filtre v metóde *deleteFilter*. Na konci *Autosizeinput* je krížik, s ktorým sa odoberú všetky filtre v metóde *deleteAll*. Kombinácia viacerých filtrov je znázornená na obrázku 11.



Obr. 11: Kombinácia filtrov

Navolené filtre sa užívateľovi zobrazia aj pri prejdení do inej časti systému alebo po odhlásení a to vďaka uloženi filtrov do *LocalStorage*. Užívatelia si môžu navzájom posielať projekty s navolenými filtrami odoslaním URL adresy. Pri každom pridaní filtra sa zavolá metóda *addFiltersToURL*. Filtre z URL sa pridajú v metóde *getFiltersFromURL*, ktorá sa zavolá len raz pri načítaní stránky a to pomocou metódy *componentDidMount*.

Nasadením filtrov do systému užívateľovi pribudol iba jeden riadok, ktorý im uľahčuje prácu a šetrí čas. Vytvoriť takéto komplexné filtre mi trvalo dvadsaťdva dní, aj napriek tomu, že vyzerajú

jednoducho. Pri programovaní tejto úlohy som využila všetky poznatky, ktoré som nadobudla pri predchádzajúcich úlohách.

5 Znalosti

5.1 Využité teoretické a praktické znalosti nadobudnuté počas štúdia

Všetky znalosti pred absolvovaním bakalárskej praxe som nadobudla až počas štúdia na vysokej škole. V predmetoch Programovanie I a Algoritmy I som sa naučila základne princípy programovania a postup akým sa riešia jednoduché algoritmy. Prvé znalosti o objektovom programovaní som získala v predmetoch Programovanie II a Algoritmy II. Pri každej úlohe som využila predmet Uživatelská rozhraní na návrh UI. Veľmi prospešným predmetom bol Tvorba aplikací pro mobilní zařízení I , v ktorom som vytvorila prvé webové aplikácie a získala základné znalosti v JavaScripte, CSS a HTML.

5.2 Chýbajúce teoretické a praktické znalosti

Pred absolvovaním bakalárskej praxe som nevedela ako viacerí vývojári pracujú na rôznych úlohach v jednom systéme. Vo firme na to používali verzovací systém Git, s ktorým som dovtedy nepracovala. Žiadne teoretické ani praktické znalosti som nemala s knižnicou React. Aj keď React využíva JavaScript, CSS a HTML, z ktorých som mala iba základné znalosti, uplatniť ich v rozsiahlom systéme bolo veľmi náročné. Novinkou boli pre mňa aj rôzne knižnice a funkcie, ktoré React umožňuje používať.

6 Záver

Počas odbornej praxe som získala veľa nových teoretických ale hlavne praktických skúsenosti. Pracovala som v tíme skúsených programátorov, ktorí mi hlavne na začiatku praxe veľmi pomáhali a snažili sa vysvetliť všetko potrebné.

Atmosféra vo firme bola príjemná, aj vďaka tomu sa mi na zadaných úlohách pracovalo veľmi dobre. Ak sa pri programovaní vyskytol nejaký problém, vždy mi poradili alebo poskytli nápovedu ako pokračovať ďalej. Za každou vyriešenou úlohou sa vykonal refaktor kódu, pri ktorom mi vysvetlili ako sa dajú niektoré časti zjednodušiť alebo ma upozornili na chybný "coding style". Všetky úlohy sú nasedené v systéme alebo testované niektorými užívateľmi. Veľa firiem ocenilo nové funkcie, vďaka ktorým sa im v systéme Projektově.CZ lepšie pracuje.

Za veľkú výhodu pokladám knižnicu React a verzovací systém Git, ktoré možno využijem aj pri iných pracovných príležitostiach.

Literatúra

[1] Interná dokumentácia firmy

[2] React [online]. Dostupné z: https://reactjs.org/docs

[3] JSX [online]. Dostupné z: https://www.reactenlightenment.com

[4] Komponenta [online]. Dostupné z: https://codeburst.io

[5] Flux [online]. Dostupné z: http://facebook.github.io/flux